```
      // Esercizio 5 del laboratorio di Informatica 2 - Programmazione di rete.
      // Sviluppare un server che si mette in attesa di connessioni su un port dato.
      // Il server accetta due connessioni in contemporanea, apre un figlio per ciascuna
      // connessione.
      // Ciascun figlio entra in un ciclo,in cui: riceve un carattere terminatore dal
      // client, riceve una stringa (max 64 caratteri) terminata dal terminatore, conta i
      // caratteri della stringa (terminatore incluso) e restituisce il risultato al
      // client. Esce dal ciclo quando riceve come terminatore il carattere 'q' e poi
      // termina.
      // Il processo padre deve attendere la terminazione dei figli, dopo che entrambi
      // sono stati chiusi si connette ad un server di terminazione sul port dato
      // (diventa quindi un client), gli spedisce il carattere 'q', attende un carattere
      // di risposta, chiude le connessioni e poi termina. Come si può vedere è molto
      // simile al precedente e la prima parte di codice è perfettamente identica.

#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <string.h>
      // Include la libreria contenente le funzioni sulle stringhe

#define SERVER "131.175.75.61"
      // Questa costante contiene l'IP del server del laboratorio di informatica.
#define PORT 35829
      // Questa costante contiene il numero del port su cui dovete restare in ascolto.
      // Inserite il valore che vi indica il server nel testo del problema.
#define END_PORT 34567
      // Questa costante contiene il numero del port tramite cui connettersi al server di
      // terminazione. Va sostituito con il valore che vi indica il server nel testo del
      // problema.
#define MAXCONN 5
      // Indica il numero massimo di connessioni che il vostro server accetta in coda.

void addr_inizialize();
      // Funzione che inizializza una variabile di tipo sockaddr_in che identifica un
      // punto terminale della trasmissione.

void main()
{
      // DEFINIZIONE DELLE VARIABILI

  int new_sd, acc_sd, end_sd;
      // Definisco tre identificatori di socket, uno per accettare la connessione,
      // uno per la comunicazione e il terzo per la connessione al server di
      // terminazione.
  struct sockaddr_in server_addr, client_addr, end_addr;
      // Definisco tre variabili indirizzo, una è per il server, l'altra per il client,
      // la terza per il server di terminazione.
  int server_len = sizeof( server_addr );
  int client_len = sizeof( client_addr );
  int end_len = sizeof( end_addr );
      // Definisco tre variabili a cui assegno la dimensione delle strutture indirizzo.
  char term;
      // Definisco la variabile in cui memorizzo il carattere terminatore.
  char stringa[64];
      // Definisco l'array di caratteri in cui memorizzo la stringa inviata dal client.
  int cont;
      // Definisco una variabile contatore per contare il numero di caratteri ricevuti.
  char conto[3];
      // Definisco la stringa in cui memorizzo il numero di caratteri dopo la conversione
      // da intero a stringa. La lunghezza massima è 3, due cifre (max 64 caratteri) ed
      // il carattere "\n" richiesto dal testo.
```

```
pid_t pid;
    // Definisco una variabile che contiene il PID dei figli.
int i;
    // Definisco un contatore per il ciclo for.
int status;
    // Definisco un intero che riceve lo stato di ritorno dei figli.
char ch;
    // Definisco una variabile carattere per inviare e ricevere un carattere.

    // INIZIALIZZAZIONE DELLA CONNESSIONE

addr_inizialize( &server_addr, PORT, INADDR_ANY );
    // "Riempio" la variabile indirizzo del server, la porta è quella indicata dalla
    // costante PORT. La costante INADDR_ANY rappresenta tutti gli indirizzi con
    // cui la vostra macchina è collegata ad una rete. Questo perchè il vostro PC può
    // essere collegato a più reti, ad esempio ad Internet con un IP e ad una rete
    // locale (LAN) con un'altro IP.

acc_sd = socket( AF_INET, SOCK_STREAM, 0 );
    // Creo un server per accettare le richieste, utilizzo un protocollo della famiglia
    // TCP/IP e di preciso il protocollo TCP.
bind( acc_sd, &server_addr, server_len );
    // Associo al canale identificato da acc_sd uno dei punti terminali (il server).
listen( acc_sd, MAXCONN );
    // Definisco la lunghezza della coda delle connessioni in attesa.
printf( "\n Sono in attesa di connessione...\n" );

for ( i = 0; i < 2; i++ )
{         // Apro un ciclo che viene ripetuto due volte per l'accettazione delle
          // connessioni.
          // Questo viene ripetuto due volte perché i figli da creare sono due.
  new_sd = accept( acc_sd, &client_addr, &client_len );
          // Mi metto in attesa di connessione, appena ho una richiesta di connessione
          // la associo al socket new_sd. Nella struttura client ottengo i dati
          // dell'altro punto terminale, nella variabile client_len la dimensione di
          // questa struttura.
  pid = fork();
          // Sdoppio il processo creando un processo figlio.
  if ( pid == 0)
  {       // Se il pid è uguale a zero sono nel figlio.

    // DURANTE LA CONNESSIONE

    recv( new_sd, &term, 1, 0 );
          // Ricevo il carattere terminatore.

    while ( term != 'q')
    {     // Se il terminatore è diverso dal carattere 'q' entro nel ciclo, altrimenti
          // lo salto. Questo test viene ripetuto dopo la fine del ciclo.
          // Il ciclo è sostanzialmente molto simile all'esercizio precedente, a parte
          // l'istruzione recv alla fine del ciclo stesso che è utilizzata per il test.

      cont = 0;
          // Azzero il contatore.
      do
      {
          // Inizio un ciclo che si ripete finchè non ricevo il carattere terminatore.
        recv( new_sd, &stringa[cont], 1, 0 );
          // Ricevo il carattere e lo inserisco in fondo alla stringa.
        cont++;
          // Incremento il contatore;
      } while ( stringa[cont-1] != term );
          // Il test di chiusura del do-while controlla se l'ultimo carattere ricevuto
```

```
            // (quello della posizione cont-1 perchè il contatore è già stato
            // incrementato) è uguale al terminatore. Se il test è vero termina il ciclo,
            // altrimenti lo ripete.
        sprintf( conto, "%d\n", cont );
            // Converto cont da intero (parametro %d) a stringa e metto il risultato
            // nella stringa conto.
        send( new_sd, conto, strlen( conto ), 0 );
            // Invio al client la stringa che rappresenta il numero di caratteri della
            // stringa ricevuto. Non c'è bisogno del simbolo "&" prima di conto perché
            // l'identificatore conto è un puntatore al primo carattere della stringa. In
            // pratica scrivere "conto2" è equivalente a scrivere "&conto[0]". Il numero
            // di caratteri da inviare è ottenuto mediante la funzione strlen, che
            // restituisce la dimensione in caratteri di una stringa. Non si può usare
            // l'operatore "sizeof()" perché questo restituirebbe la dimensione del
            // puntatore e non quella della stringa.
        recv( new_sd, &term, 1, 0 );
            // Ricevo un nuovo terminatore dal client e torno al test all'inizio del
            // ciclo. Se il test sarà verificato (terminatore diverso da 'q') il ciclo
            // verrà ripetuto.
      }     // Chiudo il while.
      close( new_sd );
            // Chiudo la connessione.
      exit( 0 );
            // Termino il figlio.
    }       // Chiudo l'if, ovvero torno nel processo padre.
  }   // Chiudo il for. Se ho già creato entrambi i figli continuo con l'istruzione
      // successiva, altrimenti ripeto dalla fork.

  wait( &status );
  wait( &status );
      // Attendo la chiusura di entrambi i figli. Il valore assunto dalla variabile
      // status non ha importanza perché non ci sono frammenti di codice per il controllo
      // degli errori.

  addr_inizialize( &end_addr, END_PORT, (long) inet_addr( SERVER ) );
      // Inizializzo la variabile contenente l'indirizzo del server, che viene convertito
      // da stringa (noi lo abbiamo scritto nel formato xxx.xxx.xxx.xxx) al formato a 36
      // bit, quello usato dalla rete.

  end_sd = socket( AF_INET, SOCK_STREAM, 0 );
      // Creo un socket per la connessione con il server terminale.
  connect( end_sd, &end_addr, end_len );
      // Eseguo la connect, passandogli il socket da utilizzare, l'indirizzo del server e
      // la dimensione di questo indirizzo.
  ch = 'q';
  send( end_sd, &ch, 1, 0 );
      // Invio il carattere contenuto nella variabile ch (ovvero 'q').
  recv( end_sd, &ch, 1, 0 );
      // Ricevo il carattere dal server.

      // CHIUSURA DELLE CONNESSIONI

  close( end_sd );
  close( acc_sd );
  close( new_sd );
      // Chiudo tutti i canali.
}

void addr_inizialize( struct sockaddr_in * indirizzo, int port, long IPaddr );
      // E' il codice della routine definita prima del main, per la spiegazione rimando
      // al libro del Pelagatti, pagina 12.
{
  indirizzo->sin_family = AF_INET;
```

```
  indirizzo->sin_port = htons( ( u_short ) port );
  indirizzo->sin_addr.s_addr = IPaddr;
}
```